COO-1018-1013

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

REPORT NO. 167

BUBBLE SCAN I PROGRAM

by

R. Narasimhan

August 18, 1964

This work was supported in part by the
Atomic Energy Commission under Contract No. AT(11-1)-1018

1. BUBBLE SCAN

1.1 Introduction

In this section we give a complete description of the first version of BUBBLE SCAN as it has been implemented in the IBM 7094-1401 system at the University of Illinois. The input-output features, the data-structure and the scanning procedure used are first described with specific reference to this implementation. However, some care has been taken to point out features intrinsic to the logical structure of BUBBLE SCAN and to differentiate them from details which are merely relevant to this particular implemented version. The second half of this section contains the detailed descriptions and schematic flow charts of MAIN and SEARCH, the two major subprograms which together make up the principal part of BUBBLE SCAN. Several outputs resulting at various stages in the processing carried out by this first version of the scanning program are included as illustrative examples to supplement the descriptions given.

1.2 Input-Output Features

BUBBLE SCAN is concerned exclusively with the scanning of one view of a bubble chamber stereotriad. In what follows this will be referred to as the input picture (as <u>the picture</u>, for short) for the scanning program. We shall assume that this input picture is digitized and suitably preprocessed. For further comments concerning preprocessing, see below under Section 2.2.

Starting with such an input picture, BUBBLE SCAN is set up to generate two output lists: (1) a TRACK LIST, and (2) a VERTEX LIST. (Here and in the rest of this paper, track and vertex are used as generic names. Only those interaction points involving two or more tracks are listed as vertices. All the rest are grouped under terminals.)

The VERTEX LIST consists of a set of VERTEX names, each vertex name pointing to a list of track names associated with that vertex. (In Section 1.3 below, we consider the data structure in greater detail and explain the actual format of these list structures as compiled in the IBM 7094.) All other tracks

in the picture not associated with any vertex are together listed by name in the TRACK LIST. Thus the two lists are mutually disjoint and, as will be explained in greater detail later, information is generally stored in BUBBLE SCAN without duplication.

Each track name and vertex name carries with it other subsidiary information for further classification of these names (at the post-editing level) into V-types vertices, stars, beam tracks, spirals and so on. This peripheral information in BUBBLE SCAN is computed according to the following scheme:

Tracks: With each track name is associated a list of points P_i (identified by their coordinates x_i, y_i) which together define the track spatially in the picture. We shall describe later how these points P_i are actually chosen. Each track name contains three kinds of additional descriptive information:

1. End point information: Interior terminals, wall terminals (which wall, etc.), vertex terminals, etc.
2. Curvature information: The total amount of turning measured in units of 45°-turns is stored together with the information whether the sense is clockwise or counterclockwise.
3. Length information: This, for the present, is restricted to a count of the total number of points listed on the track. Since our point listing is done in a systematic manner, a qualitative idea of the lengths can be easily inferred from this count.

Vertices: Vertex names carry two types of descriptive information:

1. The (x,y) coordinates of their position in the picture.
2. The associated labels which provide information about the local orientation of tracks ending at each vertex. From the number of associated labels one can readily compute the number of tracks associated with any vertex.

1.3 Data Structure

All data storage is in the form of lists and sublists with well-defined hierarchic structures. These lists are made up of four distinct cell types, each with its own specific internal structure. These are the A, B, C

and E cell types. We shall, in this section, describe their structures and functions briefly and show how the track and vertex lists are built out of them and stored within the IBM 7094 during the execution of BUBBLE SCAN.

In the IBM 7094, each cell type is represented by one or more machine words. Because of the restricted length of a machine word, it is necessary to allocate 3 machine words to each Type A cell; (these are referred to as A, A1, and A2, respectively; 2 machine words to each type C cell (C and C1); and, one machine word each to type B and type E cells. The 36 bits of each word are divided into 3 fields of 12 bits each; these are referred to as the field 1,2 and 3 respectively and denoted by writing E[1], A[2], C1[3], etc., for each of the cell-types.

All our primitive lists are linear strings and, in all our data storage, we do not use more than a 3-level hierarchic structure. At each level, a linear string is named by a HEADER. A header can be thought of as the head of a list since all links point away from the header. The cell immediately following the header in a list will be referred to as the NEAREND and the cell at the other end of the list, the FAREND. A distinct end symbol is used to terminate each linear string.

Type A cells are used exclusively as headers functioning as track names; Type C cells, exclusively as headers functioning as vertex names. Type B cells form the components (i.e., the body) of track named by an A-cell. Finally, type E cells are used exclusively by the scanning program for temporary storage of track names during the course of the processing. The specific manner of use of an E-cell will be explained later.

The respective functions served by the various fields of A, B, C, and E cell-types are shown in the schematic diagram below in Fig. 1. The significance of the WINDOW NUMBER, the specific labels used and how they are generated will be explained later during descriptions of the relevant features of BUBBLE SCAN.

It should be easy now to visualize the stored data structure at any given stage in the processing. The first configuration in Fig. 2 illustrates two tracks linked together in the TRACK LIST. The second illustrates two vertices linked together in the VERTEX LIST. The circles indicate list terminations. As was mentioned earlier, it is important to note that information

A:	TRACK DETAILS	B-LINK (NEAR END)	A LINK OR B-LINK (FAR END)
A1:	LENGTH	CURVATURE	
A2:		LABEL (NEAR END)	LABEL (FAR END)
C:	X-Y COORDINATES	A-LINK (NEAR END)	C-LINK
C2:	LABELS	LABELS	WINDOW NUMBER
B:	X-Y COORDINATES	WINDOW NUMBER	B-LINK
E:	X-Y COORDINATES	A-NAME	E-LINK

FIG. 1 CELL TYPES USED FOR DATA STORAGE

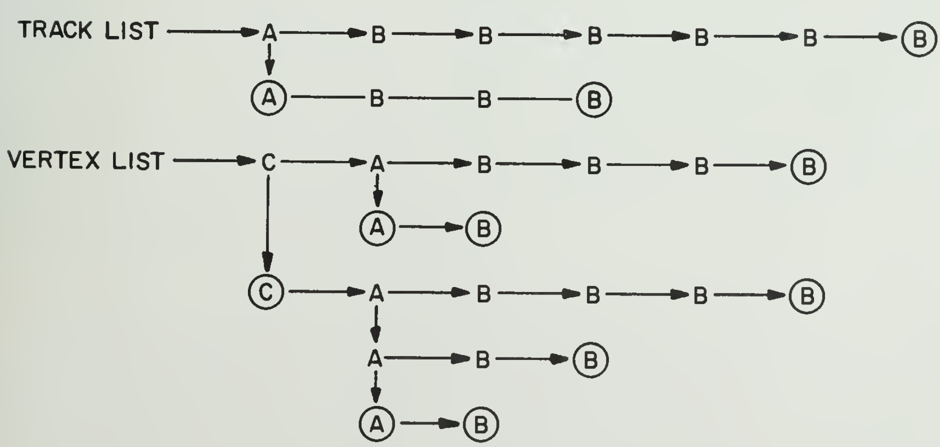


FIG. 2 LIST STRUCTURES IN WHICH OUTPUT DATA ARE COMPILED

is stored without duplication. At any given time, a particular named track will be found stored at only one place. If it has been completely processed, it will be found in the TRACK LIST; otherwise, it will be found in some E-list. If, however, a track is associated with some vertex, it will only be found in the list of A-cells associated with that vertex. Thus, the scanning proceeds transferring names of lists from one place to another, till, ultimately, all single tracks end up in the TRACK LIST, all vertex tracks in the list of A-cells associated with some one or other C-cell, and the vertices (i.e., C-cells) themselves in the VERTEX LIST.

1.4 The Scanning Procedure

In order to generate the sort of output information described in the previous section about the tracks and vertices present in the given picture, it might seem that a natural approach would be to follow through a track from terminal to terminal, or start at a vertex and compile information about its associated tracks one by one by following each through from end to end. However, this procedure has to cope with two major problems, one technical, and the other, syntactic. The purely technical problem has to do with the fact that the tracks in the input picture--except possibly for beam tracks--have essentially a random distribution of orientations. This means that in following through any single track, it is impossible to predict a priori which portions of the picture will be visited and which ones not. Hence, unless the digitized image of the entire input picture is available in the core memory of the computer all the time (thus facilitating random access to any part of it at any time), there is a good likelihood much of the processing time will be spent in transferring input data between the core memory and any auxiliary storage that is being used. This requirement also precludes the possibility of performing the scanning on-line with the digitization of the input picture, if this method of track identification is employed.

The second problem, which we have termed syntactic, is somewhat more serious. Because scanning is performed on one view at a time, ambiguities that arise when several tracks meet or seem to cross in the input picture cannot, in general, be resolved at a purely local level. There is a greater chance of being able to deal with many of these ambiguities successfully if more global information is available about the other tracks involved in the

situation (e.g., information about their lengths; curvatures; possibly, end-conditions; etc.). To be sure, certain types of ambiguities will very likely require information from all the three views for their resolution. We are assuming that these will be handled at the post-editing level and that the scanning program which we are currently describing will generate enough peripheral information to assist the post-editing program in these eventualities. In general, then, a scanning procedure is to be preferred which, while following a particular track, has available on hand partially compiled information about its neighboring tracks in the picture.

The scanning procedure actually used in BUBBLE SCAN is intended to cope with both the problems discussed above in a reasonably manageable way. Essentially the procedure amounts to scanning the input picture in a single pass from end to end and compiling information about all the tracks encountered systematically and keeping all this information continually updated as the scanning progresses. Thus a good part of BUBBLE SCAN is really concerned with elaborate bookkeeping and a relatively small part with actual tracking. We shall now describe these procedural aspects of BUBBLE SCAN in greater detail.

The digitized input picture is, to begin with, partitioned into a netword of windows of a fixed size as shown in Fig. 3. Each window represents a square raster of size 32 x 32 bits. This window size is determined by the design features of the Pattern Articulation Unit which will ultimately be used to realize parts of the algorithms incorporated in BUBBLE SCAN. The total number of windows will, of course, be determined by the fineness of digitization used, and this again depends upon the ultimate resolution required. On the basis of some preliminary estimates made using 72-inch hydrogen bubble-chamber exposures at 15:1 demagnification, it is our view that a partitioning of the picture into roughly 15 windows across and 60 windows along its length should provide adequate resolution to compile the sort of information we described earlier.

The technique used in BUBBLE SCAN is to process these windows sequentially starting with the lower left bottom and proceeding row by row (from left to right) till the top right corner is reached. Within each window we go through an iterative procedure (1) to update any of the tracks so far named and compiled which enter the window; (2) to identify and name new tracks which originate in that window and start compilations for these.

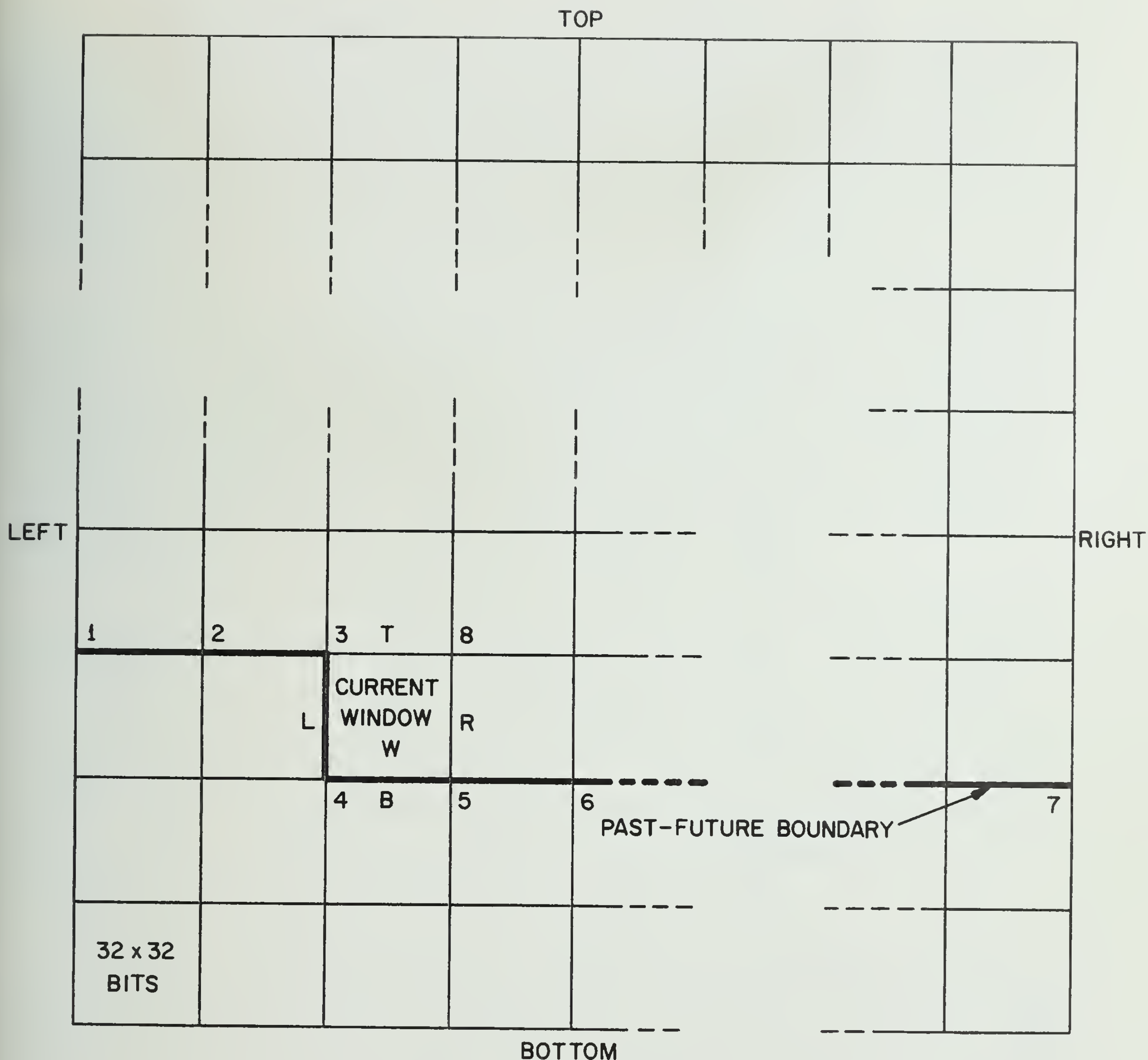


FIG. 3 PARTITIONING OF INPUT PICTURE INTO WINDOWS (SCHEMATIC)

Similar considerations hold for the vertices. Updating a track refers to listing new points which lie on the track in the window currently being processed and on the basis of this information to recomputing its length and curvature. If we go through this procedure systematically for each window, clearly, when we are finished with the last window in the picture, the compiled lists would reflect the information we set out to gather.

The scanning procedure thus divides naturally into two alternating phases: (1) an in-window phase which is primarily syntactic in character, and (2) a cross-window phase which is primarily administrative in character. Reflecting this two fold nature of the processing scheme, the program structure itself is divided into two more or less autonomous parts: an administrative part consisting of a single subprogram called MAIN and a syntactic part consisting of two subprograms called SEARCH and LABEL.

Of these three, SEARCH carries out the actual in-window processing. Given a partially completed track incident on the window, it seeks to answer the following types of questions: does the track extend into the window? If so, does it cross the window or die inside? In either case, what are the end point conditions? If two tracks meet, do they meet on a vertex? And, so on. It seeks to answer these questions both on the basis of the currently available compiled information transferred to it by MAIN and on the basis of auxiliary information generated by the subprogram LABEL.

LABEL* comprises a variety of algorithms to convert the picture inside the window into a labeled graph identifying the local directions associated with the branches (i.e., with the track segments) for subsequent classification of the nodes into vertices, bends, cross-overs, and terminals. It also identifies which nodes are associated with which walls (of the window) and which ones are interior to the window.

---

* Ultimately, the entire processing implicit in LABEL will be realized using the Pattern Articulation Unit (PAU). In BUBBLE SCAN, the algorithms which make up LABEL are written in a programming language called PAX which simulates the parallel-processing operations of the PAU.

Roughly speaking, then, for in-window processing, LABEL identifies the topological features of the portion of the picture inside the current window, and SEARCH, using this information and that supplied to it by MAIN, performs the appropriate connectivity operations to extend the tracks into and past the current window.

Finally, the MAIN program is the principal executive routine concerned with all the administrative details. It performs all the bookkeeping required for compiling and updating the syntactic information supplied to it by SEARCH, provides transfer information for cross-window references, and takes care of the general sequencing of the operations that make up the total processing system.

Figure 4 illustrates schematically the general relationship between the three processing subprograms.

1.5 Bookkeeping For Cross-Window Transfer

From our discussions in the previous section, it is clear that the MAIN program has to be responsible for two sets of bookkeeping tasks. The first, which is of a "permanent" nature, has to do with the storage of the actual data together with their classificatory information. The second, which is of a "transient" nature, has to do with providing transfer information for cross-window references. We shall describe in this section the details of the latter bookkeeping job. Since, as was pointed out earlier, this bookkeeping activity forms the main bulk of the processing done by MAIN, a description of the way this work is done will also delineate, in its essentials, the structure of the MAIN program.

Referring to Fig. 3, assume that we are just about to begin processing the window marked W. It is clear that at this moment, the boundary 1234567 divides the picture into two parts: (1) that belonging to the "past" consisting of the part below and to the left of the boundary wherein all the tracks and vertices have been identified, and (2) that belonging to the "future" consisting of the part to the right and top of the boundary wherein the naming and identification remain to be done. When the processing of window W has been completed this past-future boundary (PF-boundary) will be changed to 1238567.

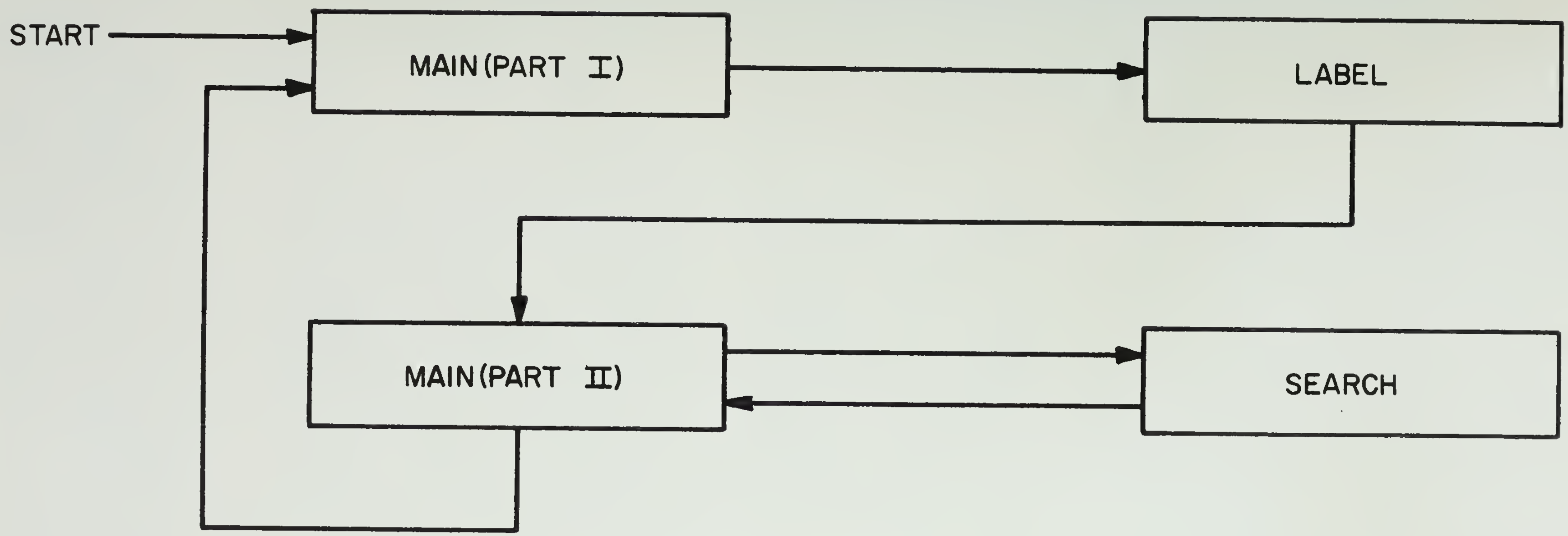


FIG. 4 BUBBLE SCAN: SCHEMATIC OF FLOW

MAIN(PART I) : STOP, IF LAST WINDOW DONE; ELSE, READ IN NEXT WINDOW AND INITIALIZE.

MAIN(PART II): TRANSFER DATA TO SEARCH AND COMPILE (OR UPDATE) ON THE BASIS OF INFORMATION SUPPLIED BY SEARCH.

It is further seen that the window W itself is bounded by two walls belonging to the past and two belonging to the future. In the figure, the first two have been marked B (Bottom), L (Left), and the latter two R (Right), T (Top). Clearly, information from the past can enter W only across B and L and can leave W to the future only across T and R. In order to optimize cross-window transfer and minimize the search involved in this process, the MAIN program maintains two lists--a B-list and an L-list--for each window and makes these available to SEARCH on entering that particular window. These lists contain the names of tracks so far identified which are incident on the current window at the walls B and L. The bookkeeping associated with these window lists is regulated by the use of a system of control counters* as described below.

With each T wall in the sequence of windows along the PF-boundary is associated a counter named V(i) TRACK, i = 1, 2, ..., M, where M is the total number of windows in any row in the partitioned picture. V(i) TRACK lists the names of the tracks incident on the ith T wall. Notice that at any given time the PF-boundary contains precisely one vertical face such as 34 in Fig. 3. A counter termed TERMINAL LEFT is assigned to this vertical face in the PF-boundary and in it are listed the names of the tracks incident on this face.

Clearly, if window W (the window just about to be processed) is the mth window in its row, its B-list should consist of precisely those elements listed in V(m) TRACK, and its L-list of precisely those elements listed in TERMINAL LEFT, at this moment. When processing of window W is complete, as we have seen already, the PF-boundary is changed to 1238567, so that the T wall of W becomes the new mth T wall in the PF-sequence and the vertical face is moved forward to the R wall of W. Since the information transferred across these wells can only come from window W, the bookkeeping procedure to be incorporated in MAIN for cross-window transfer is now well-defined and can be formulated explicitly.

---

* A counter is associated with a list of names and at any given moment points to the next name in the list. A designator would perhaps be a more precise name for such a device. However, because of certain historical accidents, the designator of the next instruction in the program sequence in a digital computer has come to be called a sequence counter. In keeping with this usage we shall retain this traditional terminology.

Three additional working counters called TERMINAL BOTTOM, TERMINAL RIGHT, and TERMINAL TOP are used by MAIN. (The last one is strictly not necessary, but its use simplifies the bookkeeping algorithms.) On entering a new window, say the mth, in a row, the contents of V(m) TRACK are transferred to TERMINAL BOTTOM. As the processing of the current window proceeds (updating the track names listed in TERMINAL BOTTOM and TERMINAL LEFT), the names of tracks that pass through (or originate in) the current window and end on its T wall are listed in TERMINAL TOP; similarly those that are incident on its R wall from inside are listed in TERMINAL RIGHT. When the processing of the current window is complete, the tracks listed in TERMINAL TOP and those listed in TERMINAL RIGHT are transferred to V(m) TRACK and TERMINAL LEFT respectively. This procedure, if systematically carried out, obviously ensures that every time a new window is encountered, its relevant L-and B-lists will be found in TERMINAL LEFT and the V TRACK corresponding to its window number (strictly the column number of the window). This window number is given by a WINDOW COUNTER. To complete the bookkeeping specification for cross-window tranfer it remains only to stipulate that no information should be transferred across end walls, i.e., window walls that coincide with the walls of the chamber.

All these cross-window transfer lists are composed of the Type-E cells described in Section 1.3 earlier. The control counters(V(i) TRACK, TERMINAL LEFT, etc.) function as headers for these lists. And because these counters together with the current content of the WINDOW COUNTER explicitly identify the window wall referenced by any given E-list, it becomes sufficient to store only the x-y coordinates of the point of incidence and the name of the track involved in the E-cells. (See Fig. 1 for the data structure.)

1.6 The Compilation Phase: Communication Between MAIN and SEARCH.

We described in the previous section the bookkeeping maintained by MAIN for crosswindow transfer and the input made available by it to SEARCH on entering a new window. It was pointed out that the in-window processing carried out by SEARCH consists in updating the old tracks incident on the current window and finding and naming new tracks and vertices that originate in the current window. The bookkeeping associated with this process is referred to as compilation and is actually carried out again by MAIN on the

basis of the information supplied to it by SEARCH. Thus, in this compilation phase, the control of the overall processing program is constantly transferred back and forth between MAIN and SEARCH; the control transfer is every time accompanied by exchange of information between these two subprograms. The dynamics of this control transfer and the communication interface between MAIN and SEARCH are described in this section.

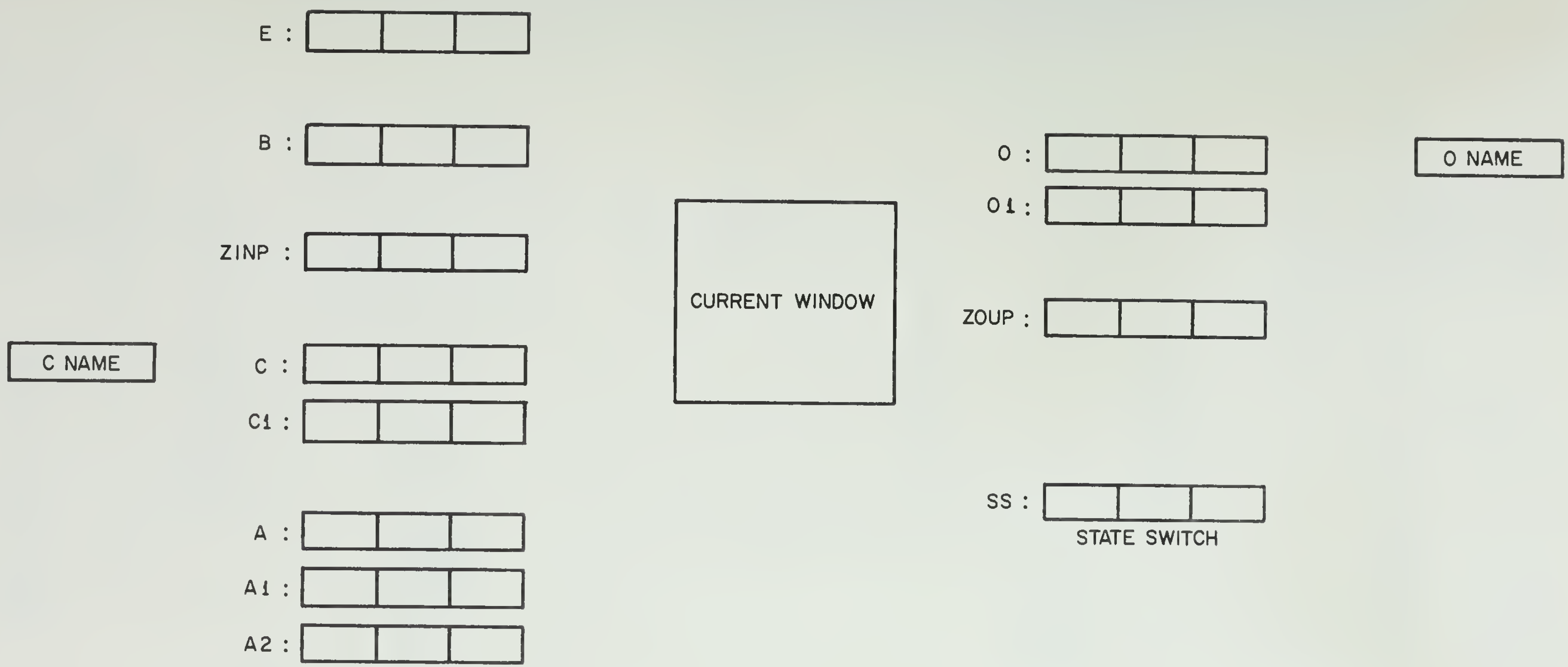
The basic logic of in-window processing is quite straightforward. It consists of a systematic and exhaustive search procedure which can be summarized as follows: first, the names listed in TERMINAL BOTTOM are updated, one at a time. Each such name refers to a track and the possible ways in which each track might behave inside the current window are all separately taken into account. As an illustration, a track incident on a B-wall might end there; or end at a terminal point interior to the window; or end on a vertex in the interior; or extend inside and up to any of the four walls. In the first two cases, this end of the track is closed. If the other end had already been closed, the compilation for this track is complete and it is transferred (from the E-list) permanently to the TRACK LIST. In each of the other cases, it is updated with the inclusion of the new track points and transferred to the relevant track counter (associated with that wall) or vertex counter. (Two additional counters called BOTTRAN and LEFTTRAN are used to list tracks that end <u>from inside</u> on the bottom and left wall, respectively, of the current window).

When all the names in TERMINAL BOTTOM have been similarly processed, the names transferred to BOTTRAN are closed and disposed of since these tracks cannot be extended any further. Next, analogous processing is carried out with TERMINAL LEFT and LEFTTRAN. This disposes of all past tracks incident on the current window. Then, new tracks <u>starting</u> from this window are identified and named, again in the order according as they start from B or L walls, the interior (C), or R or T walls. Finally, the names listed in the vertex counters are processed. This consists in checking that all the labels associated with a vertex have been accounted for, naming new tracks for the labels not yet accounted for and following them through and, finally, transferring vertices whose processing has thus been completed to the VERTEX LIST.

As shown in Fig. 5, intercommunication between MAIN and SEARCH is carried out by means of 14 cells each of which has a distinct name, i.e., a fixed address in the main storage. Each one of these cells serves a distinct function in compilation, although not all cells will be involved in every compilation. The cells marked E, B, C, C1, A, A1, A2, correspond exactly to the data cells of these types both in function and in their internal structure. In fact, these cells are used as working space to form the data cells of the respective types. The cell named O serves a dual purpose; O and O1 together function as a type-C cell in case the search ends on a vertex; O by itself functions as a Type-E cell in case the search ends on a terminal. C NAME and O NAME are two control counters which are used to name the vertices in C and O respectively. Z1NP and ZOUP are used to hold the label information associated with the starting point and terminal point, respectively, involved in the search operation. (For details of the types of label information used, see Section 1.8 further below.)

All the above are information cells. The last remaining cell termed the STATE SWITCH is a control cell which prescribes which particular compilation is actually to be carried out. The three fields of the switch are called, respectively, the INPUT STATE, the OUTPUT STATE, and the OUTPUT CODE. The sequencing at any given stage in compilation proceeds as follows: MAIN loads the proper input information cells with the relevant information and sets the INPUT STATE of the STATE SWITCH. SEARCH functions very much like a finite state machine and depending upon the input configuration (specified by MAIN) and the window configuration (prescribed by the labels), sets the STATE SWITCH to a specific OUTPUT STATE and OUTPUT CODE. The MAIN program now selects an entry, as directed by the <u>total</u> configuration of the STATE SWITCH, from a compilation table and carries out the compilation as directed in the entry. Thus the entire procedure is closely analogous to the table-directed compilation techniques used in translating artificial languages.

The INPUT and OUTPUT STATES used in BUBBLE SCAN are shown in the flow charts in Fig. 6. The states OLD TERMINAL BOTTOM/LEFT (OTR BTM/FLT) are used while processing the wall counters TERMINAL BOTTOM and TERMINAL LEFT, respectively. These counters, it will be recalled, point to E-lists which contain names of tracks incident on these walls. For these input states,



E [1] : X-Y OF INPUT POINT.
E [2] : CURRENT TRACK NAME.
ZINP : LABELS OF INPUT POINT.
C,C1 : CURRENT VERTEX DETAILS.
C NAME : NAME OF CURRENT VERTEX.
A,A1,A2 : CURRENT TRACK DETAILS.

O [1] : X-Y OF OUTPUT POINT.

ZOUP : LABELS OF OUTPUT POINT.
O,O1 : OUTPUT VERTEX DETAILS.
O NAME : NAME OF OUTPUT VERTEX.
SS [1] : INPUT STATE.
SS [2] : OUTPUT STATE.
SS [3] : OUTPUT CODE.

FIG. 5 COMMUNICATION INTERFACE BETWEEN MAIN AND SEARCH

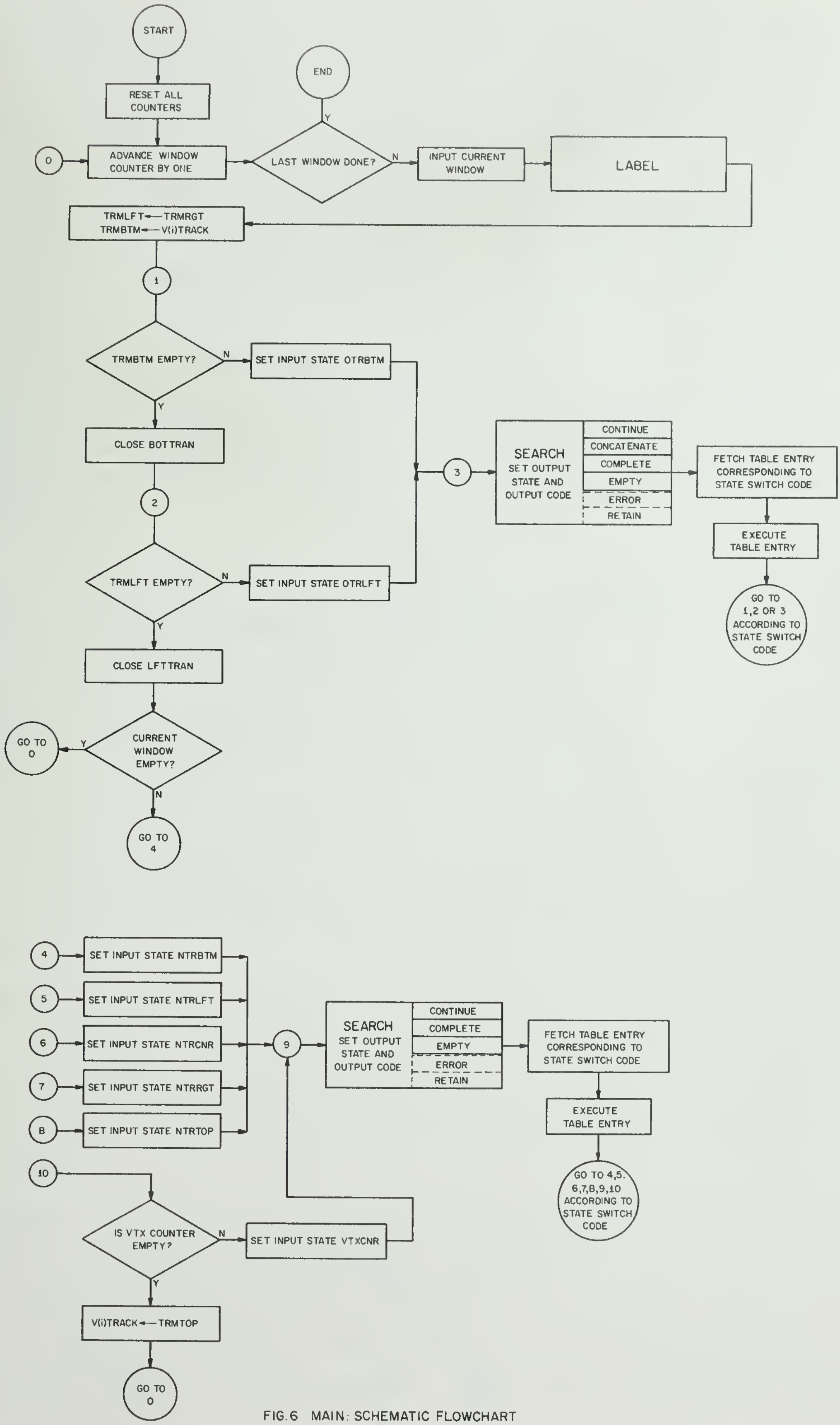


FIG.6 MAIN: SCHEMATIC FLOWCHART

the MAIN program, then, has to fill in this information in E. E[1] (i.e., field 1 of E) will now contain the coordinates of the incident point, and E[2], the name of the incident track. The header of this track is transferred to A, A1, A2 and the control is transferred to SEARCH.

The input states NEW TERMINAL BOTTOM/LEFT/CENTER/RIGHT/TOP (NTR BTM/LFT/CNR/RGT/TOP) refer to compilation for new tracks which originate in the current window from its bottom or left walls, interior, or right or top walls, respectively.

Finally, the input state VERTEX CENTER (VTXCNR) is used when the vertex counter is being processed. The header of the vertex named by the counter is brought to C, C1 by MAIN. During the vertex processing, the header of the associated vertex track currently being updated (or generated) is held in A, A1, A2.

Six different output states are employed which are identified by the names: CONTINUE, CONCATENATE, COMPLETE, EMPTY, RETAIN and ERROR. We shall now briefly explain the significance of each one of these.

CONTINUE: Normally the points which are listed in a track (i.e., the B-cell entries) are restricted to those where a track crosses a window wall. However, in certain cases, interior points are also listed when they identify well-defined bends or crossovers. In such instances, SEARCH waits in the CONTINUE state for the MAIN program to incorporate the selected point in the track list; when this is over, SEARCH proceeds with the processing of the current track.

COMPLETE: This output state implies that the processing of the current track is finished. In this case, the OUTPUT CODE (q.v.) specifies the termination details (i.e., whether the track extends to the walls, dies inside, ends on a vertex, etc.) and the appropriate compilation is then carried out by MAIN.

CONCATENATE: Because of the externally imposed order in which the windows are processed, tracks will not always be compiled from end point to end point (for comments relating to this, see Section 1.4 above). Quite often, different segments of a single track will be encountered in different windows at different stages of the processing. These segments would have been

given different names and attached to different wall counters. It is necessary to bear this in mind while processing old tracks and to piece two segments together when their end points meet. In order to be able to do this, whenever an old track extends into the window and ends on a wall, its terminal point is marked. Every time a fresh track is taken up for processing, its incident point is checked to see whether it has already been marked as an end point. If it has been marked, the corresponding track is searched out and the two segments concatenanted. (It is easy to verify that the searching need be done only among the list of names associated with BOTTRAN and LEFTTRAN; see the first few paragraphs of this section.) One of the two names is removed from the list. Because of the manner of our bookkeeping, this concatenation is a straightforward procedure to carry out.

EMPTY: The meaning of this output state varies with the different input states. In OLD TERMINALS, this implies that the current track does not extend beyond the incident wall and so should be considered as terminated. In NEW TERMINALS, this means that no additional new track starts from the specified wall. Finally, in the case of VERTEX, EMPTY signifies that all the labels associated with the current vertex have been accounted for.

RETAIN: This output state is designed to enable SEARCH to accumulate additional information to differentiate between vertices and crossovers and bends that occur inside the window, or to resolve possibly other kinds of syntactic ambiguities. In case the information available about the current track is insufficient, SEARCH stores the appropriate data concerning the situation internally and holds over processing this track till it has processed the others. In such an eventuality, the state RETAIN signals the MAIN program to remove the current track from the input and present it again at the end of the current sequence of inputs. (In the existing version of BUBBLE SCAN, the bookkeeping required for RETAIN has not been incorporated.)

ERROR: Any known anomalies or forbidden situations result in the output state ERROR. In the present version of BUBBLE SCAN, MAIN and SEARCH have their own error routines which they make use of independently. Hence, no distinct state ERROR is currently being used.

The OUTPUT CODE is set up to provide the answers to the following questions concerning the terminal point of each updating operation:

1. What type of node has SEARCH stopped on?
2. If the node is a terminal, does it lie interior to the window or on the bottom, left, right or top wall?
3. If the node is a vertex, has this vertex been visited before?

The total configuration of the STATESWITCH, then, specifies a particular entry in the compilation table. Each entry is in effect a subroutine, which making use of the parameters (i.e., what input state? what output state? what end point condition?) given in the STATESWITCH carries out the actual bookkeeping operations necessary for the updating of the lists in question.

1.7 MAIN: Schematic Flow Chart

A schematic flow chart of the MAIN program is shown in Fig. 6. It is clear from this figure that the main sequencing is almost entirely a function of the bookkeeping parameters and hence determined by the states of the various control counters. These control counters, as we saw, point to lists of track and vertex names which have relevance to the window currently being processed. When all the names referred to by all the counters have been taken care of, one major cycle is complete; the MAIN program advances the WINDOW COUNTER and proceeds to the adjacent window to repeat the entire process over again. The processing terminates when the last window in the picture has been completed.

1.8 SEARCH: Decision Logic and Schematic Flow Chart

We saw in Section 1.4 above that the in-window processing of SEARCH is based on the information generated by a subprogram called LABEL. It was pointed out there that LABEL, in effect, converts that portion of the input picture contained within the current window into a labeled graph thus identifying the basic topological features in it. Specifically, to each point in the picture is assigned one or more of four primary, directional labels: N (to signify that the point lies on a North-South road); A (to signify it lies on a Right-Diagonal road); E (to signify it lies on an East-West road); and, B (to signify it lies on a Left-Diagonal road). Points which have been assigned more than one label constitute the nodes and the rest branches:

(thus each branch represents a road-segment lying along one of four principal directions, N, E, A, B. Tracks are then identified as concatenations of road-segments subject to restrictions on what types of nodes may occur on them.)

Further labels are assigned to each of the nodes: (1) to signify whether the node is interior to the window or lies at least partially on one of the window walls; (2) to signify the directions in which roads lead away from it. These are denoted by 8 direction numbers (1,2,3,4,5,6,7,8) which pairwise correspond to the 4 direction labels: N(3,7); A(2,6); E(1,5); B(4,8).

Making use of the above labels, SEARCH classifies the nodes into four categories as follows:

(1) TERMINALS: All nodes which lie on future walls (i.e., right or top wall); to these are added the end points of road segments which cannot be continued further along that direction.

(2) CROSSOVERS: All nodes, not on future walls, which have two direction labels, and four associated direction numbers.

(3) BENDS: All nodes, not on future walls, which have two direction labels, two associated direction numbers, the angle included is either 45° or 90°, and the change in direction is in the same sense as the past compiled curvature or differs from it by only 90°.

(4) VERTICES: All nodes not classified as terminals, bends or crossovers.

The node classification, except in the case of terminals, is done when the node in question is visited the first time. At that time vertices and terminals are also marked to signify they have been visited once, so that when they are encountered a second time they are not assigned a new name over again. (This procedure also enables the concatenation of tracks whose ends meet on the current window walls.) The strategy of deferring classification of a node till this information is actually required ensures that no node is classified till all the label information that is likely to be of relevance is available to SEARCH at the time of its classification. It is for this reason also that multiple-labeled nodes on future walls (of the

current window) are not classified till they are encountered again in the adjacent windows. The current labels of these nodes are passed on to the future in the bookkeeping associated with the cross-window transfer. (See Fig. 1, in particular cell A2.)

The decision logic of SEARCH is schematically shown in Fig. 7. Thus, in the input state OLD TERMINAL, when a track is presented to it by the MAIN program, SEARCH determines which one of the following alternatives applies and carries out the appropriate action:

1) There exists no node (including terminals) at the input point; so the track does not extend into the current window. The output state is EMPTY.

2) There exists a node at the input point and it has not been visited before; SEARCH classifies it by adding together the labels transferred to it by MAIN and the labels generated in the current window. According as the node is a vertex or not, SEARCH either gives the output state COMPLETE (VERTEX), or continues the track inside till the next node is encountered. If this is a vertex or terminal, the output state is COMPLETE; else, it is CONTINUE and the track is further continued after the control returns to SEARCH, till some end condition for COMPLETE is encountered.

3) The node at the input point has been visited before. In this case, if the node is a vertex, SEARCH terminates the track by giving the output state COMPLETE (VERTEX NAMED). Otherwise the output state is CONCATENATE.

Similar considerations apply for the input states NEW TERMINALS and VERTEX, as indicated in Fig. 7.

1.9 BUBBLE SCAN: Miscellaneous Implementation Features

In this final subsection we describe some miscellaneous details that might be of interest concerning the present version of BUBBLE SCAN as it has been implemented in the University of Illinois IBM 7094-1401 System.

Memory Organization: The 7094 core memory is organized into three parts as follows: (1) the major portion of the machine memory forms the MAIN MEMORY for the program. This consists of single words individually addressable in the usual manner. (2) A second part of the core memory of size 2^{12} words, is linked together and is used as the LIST MEMORY. All E and B cells are borrowed from and returned to this memory. A special control counter called NEXTLIST is used to name the next available cell in this memory at any

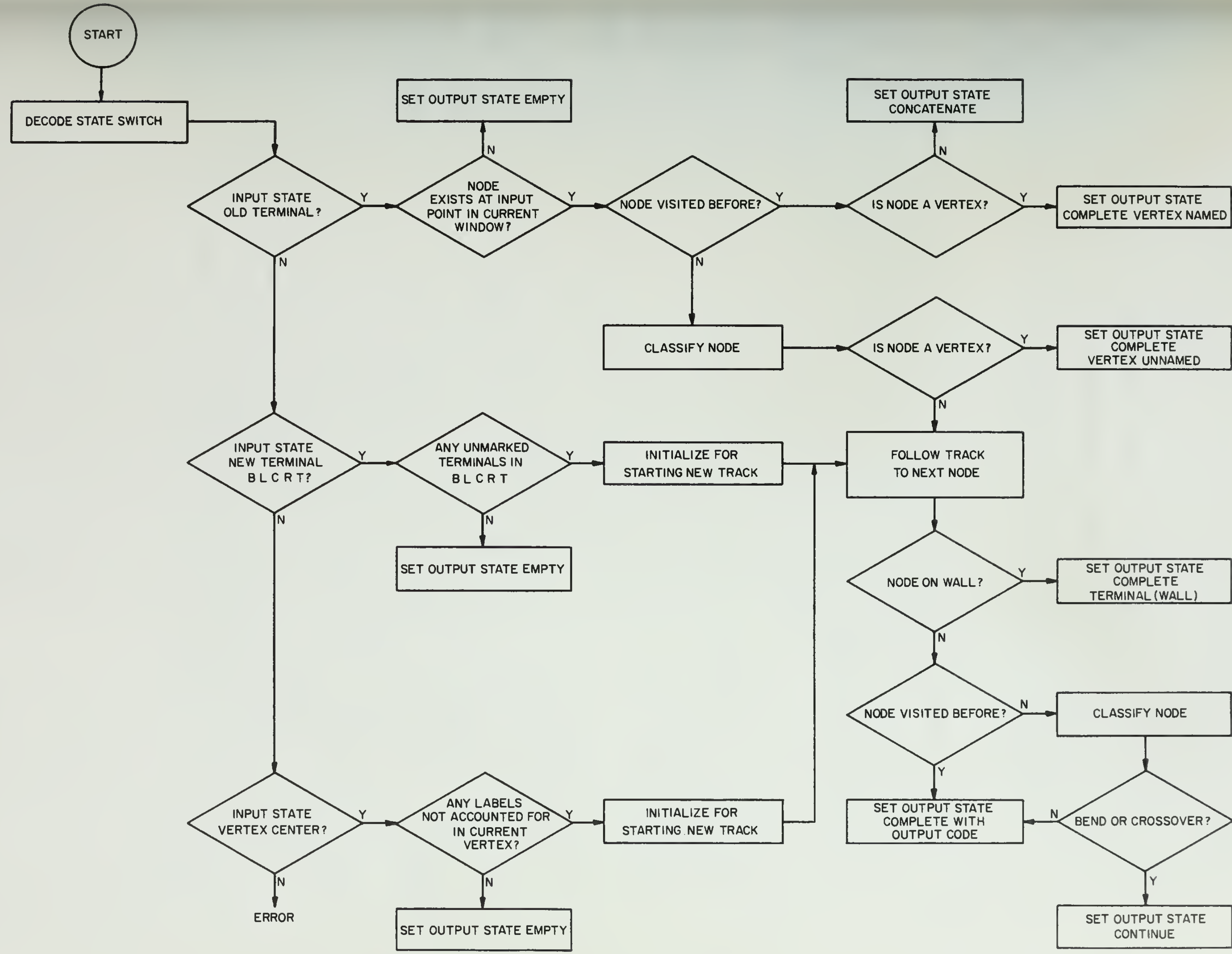


FIG. 7 SEARCH: SCHEMATIC FLOWCHART

given time. This counter is kept continually updated as cells are borrowed from and returned to this memory. (3) The last part of the machine memory forms a stack of triple words. This is referred to as the NAME MEMORY and a special counter called NEWNAME is used to point always to the next available "name," i.e., a triple. Names are borrowed out of but not returned to the Name Memory. Each name consists of three consecutive machine words. A and C cells are exclusively associated with this part of the machine memory.

Because of the restricted length of the machine words in 7094, only 12 bits are available as address fields for links in the data cells. Thus, it is necessary to address these cells indirectly. The convention employed is to refer to an item in both the LIST and NAME memories by its address relative to the base address of these memories.

Input: So far BUBBLE SCAN has only been used with test pictures consisting of only a few windows (< 20). The digitized image is prepunched on cards and input as data decks in the usual manner. For on-line testing with a scanner-digitizer, or for tests of entire 35mm frames, other arrangements possibly involving magnetics tapes will be made.

Coding: The major part of BUBBLE SCAN, including the PAX simulator, was coded in the assembly language SCATRE (relocatable version of SCAT). LABEL and portions of SEARCH were written in the PAX language. Because of their modular structure, it was quite easy to code MAIN, SEARCH and LABEL independently and test them separately using simulated input data. In fact, these routines were coded by different people at different times and the tested versions finally put together to form BUBBLE SCAN. This complete program, not including the LIST and NAME memories and the parallel processing "planes" used by LABEL and the PAX simulator occupy approximately 10,200 machine locations; of these approximately 2000 words represent the PAX simulator. Input program instructions in the PAX language are of the order of 200.

2. EVALUATION OF BUBBLE SCAN AND SUMMARY OF WORK TO BE DONE

2.1 Introduction

In the previous section we described in some considerable detail BUBBLE SCAN, the first version of a program that has been implemented on an IBM 7094-1401 System for the automatic scanning of digitized bubble-chamber negatives. As was emphasized in the introduction to that section, this program forms only a part--although a very important part--of an over-all scheme for a completely automatic analysis of bubble-chamber data. What has been established by the successful implementation of this first version of BUBBLE SCAN is the feasibility of scanning bubble-chamber negatives in a particular manner which uses efficient parallel processing algorithms at the local level to identify the topological features and, on the basis of this abstracted information, computes a global description of the input picture in a form suitable for subsequent editing and use in measurement and analysis.

From a consideration of the outputs from various stages of processing using BUBBLE SCAN, it is clear that the processing method is able to handle the input data successfully provided the digitized picture has a certain minimal definition. But to evaluate the reliability of the production version of a program operating along these lines it is necessary to take into account explicitly the actual environment in which the processing would have to be done. In the total system, BUBBLE SCAN would have to be coupled to the scanner-digitizer at the input end and to the post-editor at its output end. The reliability of functioning of this system is likely to depend on two principal factors: (1) noise in the input photograph and the noise introduced by the digitizer; (2) the nature of the information coupling required between BUBBLE SCAN and the post-editing program; the feedback paths, if any, desired; how these can be met without affecting data rates and without complicating too much the bookkeeping involved in BUBBLE SCAN. We shall consider a little more in detail both these factors in the rest of this section.

2.2 Noise in the Digitized Input Pictures

Parallel processing algorithms are extremely well-suited for combating certain types of noise at a purely local level. Gap filling and thinning routines of quite wide applicability to input pictures made up of "roads" have been developed and tested on actual digitized versions of bubble chamber pictures. Labeling techniques can be used very efficiently to connect up even large gaps in beam tracks, etc., by taking explicit advantage of their known configuration features (e.g., their predominant North-South orientation).

However, to cope with other kinds of noise (like background reflections; large, black areas, etc.) it is clear that special techniques will have to be developed. It is also clear that for optimal results, these algorithms, rather than being completely general purpose noise-cleaning routines, are best developed to cope with the outputs of specific digitizers so as to minimize the noise in the over-all system. At this stage what can be asserted on the basis of all the work we have done so far is that the programming know-how required to deal with this aspect of the problem does seem well within our scope.

2.3 The Post-Editing Program

BUBBLE SCAN assumes explicitly the use of a post-editing program to correlate the outputs from the independent scanning of the three views of a stereotriad. The main tasks of the post-editor are:

1) to rectify incorrect concatenation of track-segments by BUBBLE SCAN because of the error introduced in limiting itself to one view at a time;

2) to connect up tracks left unconnected by BUBBLE SCAN because of noise and other reasons;

3) to classify vertices and tracks on the basis of this corrected, collated information from the three views;

4) to identify the existence of events of interest in the stereotriad and specify their positions in the three views; and finally,

5) on the basis of (4), to generate an output mask in a suitable form to be made use of in the subsequent measuring phase, to obtain precise measurements of the events for further analysis.

The structure of such a post-editing program remains to be worked out. Large scale use of a first version of a post-editor on actual bubble-chamber stereotriads in conjunction with BUBBLE SCAN should enable one to determine what types of peripheral information should be compiled by BUBBLE SCAN and made available to the post-editor to render it maximally efficient in its functioning. An important aspect of this problem is determining the nature of feedback, if any, that should be present between the scanning and the post-editing programs. Specifically, it remains to be investigated whether the post-editing is best postponed till the outputs of BUBBLE SCAN from all the three views are available; or, whether it is better to have the post-editing program to function as a sort of executive routine which <u>directs</u> BUBBLE SCAN to modify its compilation strategy, as the scanning is in progress, on the basis of output editing done thus far. It is interesting to observe that if, in addition, such a post-editing executive program has the facility to switch back and forth between the three views of a stereotriad, the analogy to the on-line SMP-type analysis would become remarkably close.

It is clear that whether the post-editor, in the ultimate production version, is run as an on-line monitor of BUBBLE SCAN or not, its operational requirements are best studied by simulating such a set up. With this in view, a programming language called BUBBLE TALK is being developed for use in on-line conversation with ILLIAC III from a typewriter console. The main outlines of a preliminary version of BUBBLE TALK are given in the Appendix. As can be seen from the description given there, the programming language is being designed as a very flexible means of communication with ILLIAC III for developing, editing, and monitoring picture processing algorithms in general, and bubble-chamber data processing in particular. The facility for using light pen and CRT display as a visual input-output channel should enable one to tackle the even more fundamental problem of specifying event descriptions to the processing program. It does not seem unreasonable to expect, and to hope, that an adequately designed automatic bubble-chamber analysis program would, ultimately, accept as inputs a set of sketches <u>with marginal comments</u> of event types of interest and the exposed films, and generate from them information of interest to the physicists who designed and performed the experiments.

REFERENCES

Bond, W. D. and R. Wiegel. BUBBLE SCAN I--Part 1: MAIN. File No. 590, Digital Computer Laboratory, February 18, 1964.

Rice, R. Kevin. BUBBLE SCAN I--Part 2: SEARCH. File No. 594, Digital Computer Laboratory, April 13, 1964.

Wiegel, Roger. BUBBLE SCAN I--Part 3: LIST MEMORY. File No. 585, Digital Computer Laboratory, January 17, 1964.

Bond, W. D., R. K. Rice and R. Wiegel. BUBBLE SCAN I--Part 4: COMPLETE LISTING. File No. 606, Digital Computer Laboratory, July 1, 1964.

Narasimhan, R., J. R. Witsken and H. Johnson. BUBBLE TALK: The Structure of a Program for On-Line Conversation with ILLIAC III. File No. 604, Digital Computer Laboratory, July 2, 1964.

APPENDIX. BUBBLE TALK

A.1 Introduction

In this appendix an over-all description of the structure and scope of BUBBLE TALK as envisaged at present is given. The motivation for undertaking this work in the context of Bubble Chamber data analysis is briefly suggested here; for other related comments see Section 2.3 of the main paper. Because of the inherently complex nature of the program, it is intended to implement it in functionally self-contained stages. The appendix concludes with a summary of the current status of the programming effort in this respect.

A.2 BUBBLE TALK: Scope and Structure of the System

BUBBLE TALK, as a programming system, is primarily intended for on-line conversation with ILLIAC III concerning bubble-chamber picture processing. As a rough measure of the sophistication one would like to see achieved in the conversation, the solution of the following problem can be thought of as a representative goal: Given a roll of exposed film, and a set of sketches of desired track configurations, with suitable marginal comments concerning the features in the sketches (qualitatively in a manner analogous to the sketches of event-types given to human scanners), the program should scan through the film looking for occurrences of any of the specified configurations and print out appropriate comments; (1) in case of identification; (2) in case of partial identification (ambiguities, etc.); (3) in case of no identification. BUBBLE TALK is intended to serve as a frame work within which algorithms capable of carrying out the above task can be developed in stages tested on actual data, modified, edited, revised, and so on.

Ideally, one would like to see the conversation conducted in some such natural language as English. This, however, poses fairly formidable problems in linguistic analysis and synthesis which are not, strictly speaking, intrinsic to picture processing. Since our primary motivation at this stage is to have BUBBLE TALK serve as a versatile research tool in developing picture processing algorithms, we have tried to simplify the purely linguistic problems to a reasonable level by means of explicit built-in constraints on the permissible statement-types that can be used in the conversation.

Specifically our procedure is as follows: conversation is to be conducted using a large, but finite, "statement-type" vocabulary. Each statement consists of a single command word followed by a string of argument words admissible for that command word. Decoding (syntax-analysis) is further simplified by the liberal use of word-class brackets of different types in the body of the statement.

BUBBLE TALK functions roughly as follows: the unit of conversation, from the operator's side, is a statement. Typically, a statement may be a question pertaining to certain object types in the input picture (or, a specified portion of the picture, say, a specified window); or pertaining to some attribute of a named object of some type; or, the statement may be a command to compute the value of a certain attribute of a named object; or to locate and display a part of the picture having specified attributes; or to perform certain library routines on the display; and so on. From the point of view of the program, the conversation consists in decoding the input statement, performing the action requested and printing out a suitable reply depending on the result of the action.

Basically, then, BUBBLE TALK consists of an executive routine which plays the part of the operator's companion inside the machine as described above. It does so by making use of (1) a <u>dictionary</u> which contains the definition of all the terminology used in the conversation, and (2) a system of <u>data files</u> in which is accumulated all the information that has been abstracted out of the input picture on the basis of the conversation thus far. The data files are suitably cross-referenced and indexed for efficient retrieval of this stored information. When the operator types in a statement, the executive program decodes it and decides (by consulting the dictionary) what is being asked for; then it (1) checks the appropriate files to see whether the information asked for is already available, and if not, (2) performs the appropriate action (again, consulting the dictionary), and (3) prints out a suitable reply from a repertoire of replies admissible for this command.

As implied by this description, BUBBLE TALK is really a rather complex information retrieval program which, instead of dealing with documents and descriptions, locates objects with specified attributes in pictures, computes values of attributes of named objects, and so on.

A second aspect of the executive routine has to do with the dynamics of the on-line communication links. On-line input to the processing part of the program is from three sources:

(1) Console typewriter, $\Big\}$ from the operator
(2) Light pen on display tube,

(3) Scanner-digitizer.

Similarly, the executive routine outputs, on-line, either to the console typewriter or to the display tube.

The major feature of BUBBLE TALK which renders it a flexible medium for conversation is the facility provided for the operator to augment the dictionary according to his contextual needs, by defining new object types and new attributes in a standard format. These object types and attributes can then be used in subsequent statements in a normal manner as if they were part of the primitive vocabulary of the conversation.

A.3 Objects, Attributes and Statement Types

The primitive object types are points (of the digitized image of the picture) and the primitive attributes are black/white. New object types are defined making use of the definitions of known object types and propositional expressions involving values of their attributes. Thus, for example, road segments are defined as connected sets of black points with certain directional labels; nodes are defined as connected sets of labeled black points with multiple labels; a track is defined as a string of road segments with certain restrictions on the types of nodes that can occur on it; an event of a particular type is defined as a vertex (or a set of vertices) with associated tracks having certain attributes; and so on.

An attribute is a function defined over object types. Every object type has a set of admissible attributes. These are to be specified whenever a new object type or new attribute is defined. Particular occurrences of object types in the picture are referred to by identifiers (which are the names of these specific objects, e.g., ALPHA as the name of a particular node, belonging to the object type vertex, etc.).

With each identifier is associated a data file in which is accumulated all the information pertaining to that identifier (e.g., what object type it belongs to, its x-y coordinates, the values of its various attributes, etc.) This file is consulted in all the dialogue pertaining to this identifier.

The statement types divide naturally into several categories according as they pertain to

(1) Dictionaries: (ADD TO object type/attribute definition lists; ADD TO library subroutines; EDIT commands for deleting, inserting, replacing specified portions of dictionary-library; PRINT commands with formats, etc.)

(2) Data Files: (CREATE data files by locating and naming objects; NAME objects pointed in display; INPUT to data files by computing attribute values of named objects; OUTPUT from data files, etc.).

(3) Display: (LOCATE and DISPLAY named objects or objects with specified attributes; DISPLAY specified windows in the picture; DO library subroutines on display; STORE outputs of DO statements; NAME outputs of DO statements; DISPLAY these named outputs; etc.)

(4) Questions: (Questions concerned with occurrence; questions concerned with attributes; questions concerned with numbers, etc.)

(5) Initializing: (Pertaining to view numbers; pertaining to experiment numbers; general bookkeeping associated with scanner-digitizer, etc.).

The specific formats and other syntactic details of individual statement types are not included here.

A.4 Current Status

An initial version of BUBBLE TALK incorporating a partial list of statement types (mostly from groups 1, 2, 3, above, but excluding the use of light pen with display) has been worked out in considerable detail. This includes the specification of data-structure for the various files and dictionaries, specification of algorithms for processing-macros, and a flow chart of the executive routine using these macros. A complete account of this first version of BUBBLE TALK may be found in Digital Computer File No. 604, "BUBBLE TALK: The Structure of a Program for On-Line Conversation with ILLIAC III," by R. Narasimhan, J. R. Witsken and H. Johnson.